Certifying Year 2000 "Fixes"

Jeffrey Voas
Reliable Software Technologies

There is much less talk about certifying the correctness of year 2000 (Y2K) conversions than there is about how to make the conversions. Certifying that Y2K "fixes" were appropriate can be done easily by using a combination of different software testing techniques. This article describes these techniques and why they should be considered essential processes in any Y2K conversion solution.

The lifecycle cost to maintain software can exceed the costs to develop the original code. This is particularly true for software systems that are expected to continue in service for 20 to 30 years, in which maintenance can account for 50 percent to 70 percent of the total lifecycle costs for a software system. Of these maintenance costs, testing can account for 50 percent or more of the costs [1]. Thus, the cost to test modified code can be a substantial portion of the total costs of keeping legacy code alive.

These factors hold true for Y2K fixes. To eliminate a system's Y2K problem requires that you identify what parts of the software need modification, make the conversions, and test the conversions. According to [4], more than 60 percent of all Y2K costs will go to testing. This figure can be as high as 70 percent for some projects [5].

In a regular system's lifecycle, the maintenance phase involves two key activities: fixing faults in existing code and adding new functionality to existing code. Y2K conversions can be placed into either category, depending on your perspective and the age of the system. If you view Y2K problems as fixing programmer mistakes, your fixes would be considered fault eradication. (This makes sense if the systems were built fairly recently.) If you view the Y2K problem as software that outlived its intended lifespan, you view Y2K conversions as adding functionality to aged systems. (This makes sense for systems that incorporate code created many years ago.)

Regardless of how the Y2K issue is viewed, modified code should be tested, and unmodified parts of the system should be retested to ensure that each "fixed" system is Y2K immune. As already stated, these testing expenses can be pricey. One reason is that few testing tools are smart enough to automatically know how to minimize testing costs for modified code. However, you would think a "smart" tool could determine exactly what code needed to be retested. It would be great if such an automated tool existed to distinguish that kind of code in a "optimized" mode, i.e., determine the least amount of code that needed to be retested to demonstrate that a code conversion was correct.

Unfortunately, no such tool exists. This suggests that there is a serious need for tools that seamlessly integrate with Y2K conversion tools and that test Y2K conversions. If such tools existed, the total global cost of the Y2K problem could be reduced while still providing sufficient confidence that Y2K conversions were correct. This could add up to astronomical savings, as the world-wide cost for fixes alone is $600 billion, not to mention legal liability costs that could exceed $1 trillion [2, 3].

Following is a description of what testing tools must accomplish under any scenario to provide the appropriate levels of confidence. In short, it is something like a checklist of testing processes that certify Y2K compliance.

Coverage Testing

The absolute minimal requirement is that modified code be tested (commonly referred to as "exercised"). If modified code is not exercised, it is not possible to know what its behavior will be. To exercise code, generate test cases that execute the conversions, then employ simple coverage analysis to analyze whether modified statements are hit. This can be done easily if

- The conversion tool places comments in the converted code.
- The coverage tool's parser looks for those comments.
- The coverage tool then places instrumentation to record when those statements are exercised.

Once coverage testing is successfully completed, you know that all code modifications have been executed at least once.

But from a quality perspective, it is imperative to recognize that it is barely sufficient to merely reach statements, because there are other forms of coverage testing, e.g., dataflow, that are better at fault detection than statement testing. For example, dataflow testing would allow you to test "all uses" of the year fields that were modified. Or if you attacked your Y2K problem by adding complex conditions, e.g., changing

```
if y1 < y2 then
    years_apart = |y2 - y1|
```

to

```
if (((y1 < y2) and (y2 < 00)) or ((y1 >= 00) and (y2 >= 00))) then
    years_apart = |y2 - y1|
else if (y1 <= 99) and (y2 >= 00) then
    years_apart = (99 - y1 + 1) + (y2 - 00)
```

to avoid increasing the size of year fields, you should use a coverage testing approach like multiple-condition coverage

(MCC) or condition-decision coverage (C/DC). Note that dataflow, MCC, and C/DC coverage testing are more thorough than statement testing.

However, coverage testing is only one ingredient in Y2K certification. After all, even complete coverage testing of all code modifications does not imply that all Y2K conversions are correct. All fixes could be correct, but the fixes may have broken system functionality; that is, all the year fields may now work properly, but other functionality that used to work now does not. (Such a situation could occur because of a lurking fault that could not be triggered until a year greater than 1999 is used.)

This potential problem needs to be mitigated by retesting existing functionality. To do so, regression testing can be employed, which is the next ingredient needed for Y2K certification.

Regression Testing

Regression testing employs a suite of test cases (usually with respect to the requirements or specification) to ensure the outputs from the original code and converted code are identical for each member of the suite. Note here that it is assumed that for each member of the suite, there should be identical behavior for both versions of the code. Regression testing provides evidence that the conversions have not affected any functionality that should have remained unaffected.

System-Level Testing

But there will also be system-level inputs that we would want to result in different outputs (between the converted and unconverted programs). If this were not the case, why was the software converted? To determine that the new version is doing what you want for these inputs, employ system-level testing, which will employ test cases that represent events beyond 1999. System-level testing will serve as our last ingredient in Y2K certification.

It is important to note that system-level testing is neither a substitute for coverage testing nor does coverage testing replace system-level testing. It is possible to exercise all code conversions and not discover that a conversion fails in the context of a test that represents an event after 1999. Likewise, it is possible to system-level test with a wide variety of post-1999 scenarios that do not exercise all modifications. Thus, both forms of testing are needed for Y2K certification.

Summary

To certify that code is Y2K compliant, three different forms of testing should be employed:

- Coverage testing to exercise fixes.
- Regression testing to see whether new code breaks other system requirements that are not related to calendar dates.
- System-level test to see how the new system handles events past 1999.

These techniques do not guarantee that the conversions will integrate into your system and work correctly under all scenarios. Except for exhaustive testing, testing can never make such guarantees. Instead, testing provides confidence. And for legacy Y2K systems, that is what is needed after these "tried-and-true" systems are upgraded to handle events after 1999.

Since testing can account for 50 percent of maintenance costs, if you were planning to spend X dollars on conversion, the additional certification costs could double your cost to 2X. But without taking these defensive measures, you could get fooled into thinking that your Y2K problem is behind you when it is not. Given that you have taken proactive measures to solve your Y2K problem, this is a situation you will want to avoid.

I have deliberately simplified the Y2K certification process down to a handful of traditional testing approaches. Admittedly, there are more advanced certification processes that could be employed to provide similar results. Given that much Y2K conversion is ongoing without assurances that the conversions are correct, it should be more beneficial for practitioners to layout the basic needs for Y2K certification than to layout a Utopian "pipe dream," such as proving beyond all doubt that the legacy system is correct. ◆

About the Author

Jeffrey Voas is a co-founder of and chief scientist for Reliable Software Technologies and is currently the principal investigator on research initiatives for the Defense Advanced Research Projects Agency and the National Institute of Standards and Technology. He has published over 85 refereed journal and conference papers. He co-wrote *Software Assessment: Reliability, Safety, Testability* (John Wiley & Sons, 1995) and *Software Fault-Injection: Inoculating Programs Against Errors* (John Wiley & Sons, 1997). His current research interests include information security metrics, software dependability metrics, software liability and certification, software safety and testing, and information warfare tactics. He is a member of the Institute of Electrical and Electronics Engineers and he holds a doctorate in computer science from the College of William & Mary.

Reliable Software Technologies
21515 Ridgetop Circle, Suite 250
Sterling, VA 20166
Voice: 703-404-9293
Fax: 703-404-9295
E-mail: jmvoas@rstcorp.com

References

1. Myers, G., *The Art of Software Testing*, John Wiley & Sons, 1979.
2. "Year 2000 Prophet Preaches $600 Billion Digital Fix," *Computer News Daily*, Oct. 1, 1997.
3. Hassett, D., "Frequently Asked Questions About the Year 2000 Problem," available at http://www.y2k.com/legalfaq.htm.
4. Tech-Beamers, "White Paper Year 2000 Focus On Testing," May 10, 1996, available at http://www1.mhv.net/~techbmrs/tstgwp.htm.
5. Scheier, R. L., "Year 2000: Testing Can't Wait," *Computerworld*, Oct. 20, 1997, available at http://www2.computerworld.com/home/online9697.nsf/All/971020test.